MILITRAN

• • • a technical summary

SYSTEMS RESEARCH GROUP, INC.

A SUBSIDIARY OF GULTON INDUSTRIES, INC.

1501 Franklin Avenue

Mineola, New York 11501

What is MILITRAN?

MILITRAN is a simulation-oriented, general purpose language developed by SYSTEMS RESEARCH GROUP, INC. under the sponsorship of the Office of Naval Research. The primary purposes of MILITRAN are:

1. To facilitate the design of a simulation model to be implemented on a digital computer,
2. To reduce the time and expense necessary to program the simulation model, and
3. To assist in the extraction of meaningful results from the simulation.

The structure of the MILITRAN language is grounded in a methodology common to all simulations and thus offers the analyst a natural vehicle for the expression of his model. This same language lends itself to straightforward programming techniques without the need for elaborate circumlocutions.

Every effort has been made to make the language natural from the point of view of both analyst and programmer. For example, matrices and other arrays may be indexed explicitly on the name of an object; SPEED (THRESHER) and SPEED(SEA WOLF) correctly specify the appropriate entries of the vector SPEED without the assignment of artificial numerical values to the submarines named. Arrays having up to ten dimensions are permitted, thus eliminating the need for artificial partitioning to meet language limitations. Names of parameters may be up to sixty characters long; for example, NUMBER OF SUBMARINES IN THE NORTH SEA is a perfectly valid parameter name, easily understood by both programmer and analyst. Cryptic abbreviations are no longer needed. This flexibility, built into the MILITRAN language, reduces the difficulties associated with analyst/programmer communication; the program follows naturally from the analyst's model and the analyst can easily interpret and check the actual program statements.

All of the general processing and computational features familiar to users of FORTRAN, JOVIAL, and other algorithmic languages are present in MILITRAN *in addition to the special simulation features*. Simulations often contain a great deal of straightforward computation and data handling, and to provide special features at the expense of general power would be robbing Peter to pay Paul. Some users have stated that they find MILITRAN, because of its great flexibility, ideal for non-simulation work involving purely computational programming. The primary purpose of MILITRAN's general programming features remains, however, the reduction of effort required to program *simulations*. The following pages are devoted to discussions of specific MILITRAN features.

Objects

In defining the elements of a simulation model, the analyst frequently finds it convenient to refer to abstract objects (submarines, missiles, passengers, telephone lines, etc.) as well as to purely numerical parameters. These objects may further be grouped into classes, associated with other objects, or have numerical characteristics describing them. A conventional programming language would either force the analyst to specify his model in an unnatural way or compel the programmer to develop a scheme of artificial numerical codes, flags, and indices. MILITRAN allows any natural system definition to be programmed directly.

In addition to the usual numerical and logical variables, the MILITRAN programmer may use PROGRAM OBJECT variables to manipulate abstract objects. The two statements below are analogous:

```
ATTACKER = SEA WOLF
I = 5
```

The first statement assigns the value "5" to the numerical variable "I" while the second assigns the value "SEA WOLF" to the PROGRAM OBJECT variable "ATTACKER." PROGRAM OBJECT variables may be used with essentially the same freedom as numerical variables—they may be compared, used in logical conditions, placed in vectors and other arrays, used as subscripts or indices, formed into lists, etc. The MILITRAN statements below illustrate some of these operations.

```
OBJECT PLYMOUTH(2), RAMBLER(3), CADILLAC(1), ROLLS(2)
```

defines eight cars as basic objects in the system.

```
CLASS (LUXURY) CONTAINS EACH*CADILLAC, EACH*ROLLS
```

defines a class or group containing the one CADILLAC and the two ROLLS. The class is called LUXURY.

```
PROGRAM OBJECT MY CAR
```

defines MY CAR as a program object variable.

```
MY CAR = RAMBLER(1)
```

assigns the value "RAMBLER" to the program object variable, "MY CAR." In particular, MY CAR is now the first of the three Ramblers defined above.

```
IF (MY CAR .IN. LUXURY), HOME JAMES
```

tests the current value of MY CAR. If that value is a member of the class LUXURY, control will transfer to the statement labelled "HOME JAMES."

Numerical values may be associated with either objects or classes. Given the definitions

```
CLASS (ECONOMY) CONTAINS EACH*RAMBLER, EACH*PLYMOUTH
CLASS (CAR) CONTAINS EACH*LUXURY, EACH*ECONOMY
INTEGER HORSEPOWER(CAR), SPEED(CAR)
```

we could immediately refer to the values of SPEED(MY CAR) and HORSEPOWER(MY CAR) without resort to artificial indexing schemes.

List Processing

In the design of a simulation model it is often convenient to think of groups of related data as entries in a list. For instance, the position, assigned target, and weapon load of an aircraft in flight might constitute an entry in a "strike" list. So many similar groupings occur in the average simulation model that list processing features are especially helpful in a simulation language.

MILITRAN provides an extensive list processing capability. The statements provided for defining and processing lists are both flexible and powerful. Lists may be defined by LIST statements. The number and types of items included in a single list entry is entirely programmer-defined; numerical, logical, and program object information may be combined in any way within a list entry. List entries may be created by PLACE and PLACE ENTRY statements; modified by REPLACE and REPLACE ENTRY statements; located by the system functions MINIMUM INDEX and RANDOM INDEX; and destroyed by means of REMOVE and REMOVE ENTRY statements. The entry or entries to be modified or removed by a REMOVE or REPLACE may be specified by logical conditions of any complexity on any or all components. This permits complex updating processes to be accomplished by a single MILITRAN statement.

The flexibility of MILITRAN list processing is best demonstrated by example:

```
LIST OPERATING ( (VEHICLE, DESTINATION, PASSENGERS), CAR)
PROGRAM OBJECT VEHICLE, DESTINATION
INTEGER PASSENGERS
```

The above statements define a list and indicate the nature of information stored in each component of its entries. The use of "CAR" as a dimension indicates that the maximum number of entries possible is the number of objects in the class "CAR."

Operations on the above list might include some of the following:

```
PLACE (MY CAR, CHICAGO, 3) IN OPERATING
```

records the status of a car carrying three passengers to Chicago.

```
REPLACE (MY CAR) BY (*, NEW YORK, *-1) IN OPERATING
```

updates the status of that car after dropping one passenger.

```
REPLACE (, , * . L . 6) BY (* , * , * + 1) IN OPERATING
```

adds one passenger to *every* car in the list which is now carrying less than six passengers.

```
REMOVE (, LOS ANGELES) FROM OPERATING
```

causes *every* entry whose DESTINATION is "LOS ANGELES" to be removed from the list.

```
REPLACE ( ) BY (* , WASHINGTON , *) IN OPERATING
```

resets the DESTINATION component of *every* entry to the value "WASHINGTON."

Events

The central dynamic feature of most simulation programs is the processing of simulated events occurring either at regular intervals or at critical points in time. MILITRAN provides a convenient and flexible means of implementing both "time-step" and "critical point" algorithms. Full control of event structure is retained by the user, and automatic features may be overridden without system alterations.

The basic defining statement in a MILITRAN event is the EVENT statement. For example,

```
CONTINGENT EVENT ARRIVE ( (ARRIVAL TIME, ARRIVING CAR), 8)
```

defines the beginning of a block of statements which perform the processing associated with the arrival of cars at a given point. In addition, the same statement defines a list of parameters which distinguish one occurrence of the event from another. An END statement defines the end of the event processing block. Any number of events may be included in the program, and the processing associated with each event may take any form. At each occurrence of the event, the system clock (TIME) is automatically updated. Similarly, the entry containing parameters relating to the particular occurrence is located by the system variable INDEX. These automatic updating processes are triggered by the NEXT EVENT statement.

The selection of events in their proper simulated order is accomplished as follows:

1. By convention, the first component of each entry in an event list is considered to be the time at which the event will occur.
2. Upon execution of the NEXT EVENT statement, all event lists are examined to find the entry having the smallest time component greater than or equal to TIME.
3. TIME is set to the value of the chosen entry's time component; INDEX is set to the position of that entry within its list.
4. Control is transferred to the statement block associated with the list containing the chosen entry.

Several implications arise directly from this convention. First, every entry in every event list represents the potential occurrence of an event. Events may be scheduled or cancelled by means of simple list processing statements. Any event may cause or inhibit any other event by this means. Since the selection process is initiated specifically by the NEXT EVENT statement, several events may share blocks of coding. An unconditional transfer of control from one event to another does not violate system rules. The system clock (TIME) may be altered by the user if that is desired, and no disruption of the automatic event sequencing will result. Additional degrees of freedom are provided by the PERMANENT EVENT statement and modified NEXT EVENT statements. These permit planned variations in the "natural" event sequence.

General Features

Although much of the processing involved in a simulation program appears similar to that used in strictly computational codes, significant differences of a general nature occur. These differences must be considered in designing a simulation language, and have been an integral factor in the development of MILITRAN. We have already mentioned several general features: sixty-character identifiers, ten-dimensional arrays, and the use of objects as subscripts. Each of these features contributes heavily to the elimination of artificial coding "tricks" and to increased communication between programmer and analyst.

The iterative processes in a simulation program often involve incrementation and termination criteria which cannot be expressed in the usual algorithmic language. The MILITRAN DO-loop form allows modification of termination criteria, increment values, and even the index itself *within the iteration*. Further, exit from the loop may be made at any point without loss of current values, and indices are defined *even after normal exits*. A second form of the DO statement permits iteration over all members of a class to be specified in a convenient and concise manner. Both forms of DO-loop may be nested to any depth.

Several MILITRAN features reduce considerably the need for recompilation of a program: full diagnostics are produced by a single compiler run; the processor accepts mixed-mode expressions wherever contextual meaning is clear, thus eliminating recompilation due to missing (and unnecessary) decimal points; allocation of storage at running time eliminates recompilation for adjustment of dimensions.

MILITRAN provides extensive cross-referenced compiler listings, and further aids self-documentation efforts by permitting comments at any point . . . even within a statement.

Further Information

Three MILITRAN manuals are available to qualified requesters through the Defense Documentation Center (DDC), and are offered for sale through the Clearinghouse for Federal Scientific and Technical Information (CFSTI), Springfield, Virginia 22151. The manuals and their prices from CFSTI are:

1. *MILITRAN Reference Manual* (AD 601-794), $2.25
2. *MILITRAN Operations Manual for IBM 7090/94* (AD 601-795), $2.00
3. *MILITRAN Programming Manual* (AD 601-796), $3.50

Other inquiries concerning MILITRAN should be directed to the Special Projects Department,

SYSTEMS RESEARCH GROUP, INC.
1501 Franklin Avenue
Mineola, N. Y. 11501

MILITRAN Statements

Environment Declarations
REAL $n_1(i_1, i_2, \ldots, i_k), \ldots, n_m(i_1, i_2, \ldots, i_j)$
INTEGER $n_1(i_1, i_2, \ldots, i_k), \ldots, n_m(i_1, i_2, \ldots, i_j)$
LOGICAL $n_1(i_1, i_2, \ldots, i_k), \ldots, n_m(i_1, i_2, \ldots, i_j)$
OBJECT $n_1(i_1), n_2(i_2), \ldots, n_m(i_m)$
PROGRAM OBJECT $n_1(i_1, i_2, \ldots, i_k), \ldots, n_m(i_1, i_2, \ldots, i_j)$
CLASS (c) CONTAINS $a_1, a_2, \ldots, a_m$
NORMAL MODE $m_1(a_1, a_2, \ldots, a_k), m_2(b_1, b_2, \ldots, b_r)$
VECTOR N ($(a_1, a_2, \ldots, a_i), d_1, d_2, \ldots, d_i$)
COMMON $n_1, n_2, \ldots, n_i$

Substitution Statement
a = b

Control Statements
GO TO s
PAUSE j
STOP
IF (b), s_t, s_f
UNLESS (b), s_f, s_t
DO (s) UNTIL b, $n = e_1, e_2$
DO (s) FOR a . IN . b
CONTINUE

List Processing Statements
LIST n ($(c_1, c_2, \ldots, c_i)$, d)
LENGTH (n)
RESET LENGTH (n) to p
PLACE $(e_1, e_2, \ldots, e_i)$ IN n
REMOVE ENTRY n(k)
PLACE ENTRY m(j) IN n
REPLACE ENTRY n(k) BY $(e_1, e_2, \ldots, e_i)$
REPLACE ENTRY n(k) BY ENTRY m(j)
REMOVE $(b_1, b_2, \ldots, b_i)$ FROM n
REPLACE $(b_1, b_2, \ldots, b_i)$ BY $(e_1, e_2, \ldots, e_i)$ IN n
REPLACE $(b_1, b_2, \ldots, b_i)$ BY ENTRY m(j) IN n
MINIMUM INDEX $(n(b_1, b_2, \ldots, b_i, b_x), s)$
RANDOM INDEX $(n(b_1, b_2, \ldots, b_i, b_x), s)$

Event Statements
PERMANENT EVENT n($(a_1, a_2, \ldots, a_i)$, d)
CONTINGENT EVENT n($(a_1, a_2, \ldots, a_i)$, d)
NEXT EVENT
NEXT EVENT $(n_1, n_2, \ldots, n_i)$
NEXT EVENT EXCEPT $(n_1, n_2, \ldots, n_i)$
END
END CONTINGENT EVENTS (s)

Procedure Statements
PROCEDURE n
PROCEDURE $n(a_1, a_2, \ldots, a_n)$
EXECUTE n
EXECUTE n $(a_1, a_2, \ldots, a_n)$
RETURN
RETURN a

Input-Output Statements
FORMAT (Format Specification)
READ (t, s) List
WRITE (t, s) List
READWRITE (t_1, s_1, t_2, s_2) List
BINARY READ (t) List
BINARY WRITE (t) List
END FILE RETURN (s)
END RECORD RETURN (s)
BACKSPACE (t)
BACKSPACE FILE (t)
END FILE (t)
REWIND (t)
UNLOAD (t)

SYSTEMS RESEARCH GROUP, INC.
1501 FRANKLIN AVENUE
MINEOLA, NEW YORK

MILITRAN CODING FORM

Identification
EXAMPLE
73 80

Job Finite Length Queue
Coder Zi
Date
Page 1 of 1

STATEMENT LABEL (1–6)	Cont. (12)	MILITRAN STATEMENT (13–72)
		READ (5, INPUT) ARRIVAL RATE, SERVICE RATE, DAYS PER RUN,
	1	PROFIT PER TRUCK, COST PER SPACE PER DAY
INPUT		FORMAT (5F10.3)
		PLACE (DAYS PER RUN) IN END OF RUN
		PLACE (-LOG(RANDOM)/ARRIVAL RATE) IN ARRIVAL
		NEXT EVENT ... SIMULATION BEGINS
		CONTINGENT EVENT ARRIVAL ((ARRIVAL TIME), 1)
		IF(LENGTH(SERVICE QUEUE).GE.SPACES), NEXT TRUCK
		SERVE TIME = MAX(TIME, SERVE TIME) - LOG(RANDOM)/SERVICE RATE
		PLACE (SERVE TIME) IN SERVICE QUEUE
NEXT TRUCK		REPLACE ENTRY ARRIVAL(1) BY (TIME-LOG(RANDOM)/ARRIVAL RATE)
		END ... THIS STATEMENT IS ALSO INTERPRETED AS 'NEXT EVENT'
		CONTINGENT EVENT SERVICE QUEUE ((SERVICE TIME), SPACES)
		PROFIT = PROFIT + PROFIT PER TRUCK
		REMOVE ENTRY SERVICE QUEUE (INDEX)
		END
		CONTINGENT EVENT END OF RUN ((END TIME), 1)
		WRITE (6, OUTPUT) PROFIT - SPACES*TIME*COST PER SPACE PER DAY
OUTPUT		FORMAT (9H PROFIT = F7.2)
		STOP
		END

MILITRAN Features

Event Processing

- Programmed alteration of natural event sequence can be specified at any time.
- Permits any timing convention without system modification.
- Attacker, target and critical parameters located automatically.

Object/Class Definition

- Unlimited association of objects within a class structure.
- Automatic association of data with objects.
- Free manipulation of object identifiers as variables.
- Input/output of object identifiers.

List Processing

- List structure permits several components to be associated directly.
- A list may be processed as a whole; list entries may be accessed with or without reference to objects.
- Full capability to place, remove, replace, or modify list entries; entry selection by position or any set of logical conditions on any or all components.

Arrays

- Number of dimensions in an array limited only by storage capacity. Ten-dimensional arrays are entirely possible.
- Dimensions may be expressions involving parameters to be supplied at load-time.
- Subscripts may take the form of any expression and may be nested to any depth.

General

- Identifiers may be up to 60-characters in length.
- DO-loops are controlled by logical expressions; indices may be altered within the loop; transfers to and from loops are not restricted.
- Debugging time is reduced to a minimum by a full set of diagnostics and a complete cross-reference system.
- All levels of diagnostics are issued in a single compiler run.

SYSTEMS RESEARCH GROUP, INC.

A SUBSIDIARY OF GULTON INDUSTRIES, INC.

1501 Franklin Avenue

Mineola, New York 11501